Journal of Statistical Software

June 2012, Volume 49, Issue 11. http://www.jstatsoft.org/

Oscars and Interfaces

Antony Unwin
University of Augsburg

Abstract

Graphical user interfaces (GUIs) are gradually becoming more powerful and more accepted. They are the standard way of interacting with the web and play an increasing role in many software applications. Nevertheless, they have not been generally adopted, and critics point to particular weaknesses and disadvantages. Many of these are due more to flaws in design and implementation than to the basic concepts of GUIs. More attention could be paid to what users want to do and how a GUI might be developed to support these goals. Using a dataset about Oscar nominees and winners, this paper considers what analyses statisticians might carry out and what kind of GUI would be appropriate for these tasks. (It also offers some insights into the Oscars dataset.)

Keywords: GUI, **iplots**, **JMP**, **Mondrian**, Oscars.

1. Introduction

Graphical user interfaces (GUIs) have been around for over twenty years. Some are good, some are bad, and some you just get used to, appreciating their good points and working around their bad ones. GUIs are liked by users who want a simple, intuitive interface, which allows them to do most of what they want to do without fuss and complicated commands. The best GUIs work in a way which relates to how we think. For instance, to move a file from one folder to another we can pick it up (select it in software terms) and move it. GUIs are disliked by more demanding users who want greater power and control and are prepared to master complex command structures to be able to accomplish precisely what they want to do. In an ideal world we would have the advantages of both. Beginners and casual users are more likely to want GUIs, while expert and regular users are more likely to prefer command-line interfaces (CLIs). It is particularly attractive with CLIs that you can repeat procedures exactly the way you have performed them before. And that, of course, is a key feature of the scientific method: reproducibility. Where GUIs score is in exploratory analyses, situations where flexibility and immediacy are more important than tight control and precision.

Amongst statistics software packages, two stand out as having developed impressive GUIs early on, **DataDesk** (Velleman 1997) and **JMP** (SAS Institute Inc. 2012). It is no coincidence that both started on Macintosh computers. **JMP** is still being enhanced and improved and offers a wide range of statistical tools. Its basic design structure has not changed since its beginnings in the late 1980's and any discussion of its merits or demerits should bear in mind that work on it was started at a time when very different computing environments prevailed to those we are used to to-day. The fact that it still looks modern pays testimony both to the soundness of the original design and to the lack of progress by other packages. Some packages have added front ends of menus and dialog boxes to their modules, providing a limited GUI experience. Within R (R Development Core Team 2012) there is a number of different groups working on GUIs, as can be read in this issue or on the R GUI webpage (Grosjean 2010).

Through the ever-increasing presence of the web in our everyday lives, we are all becoming familiar with GUIs for various basic tasks, be it banking, booking flights or buying books. It seem highly probably that GUIs will get better, and more and more people will come to expect them, even for complex tasks, such as statistical analyses.

2. What is a GUI?

The acronym GUI is pronounced "gooey", which means sticky and syrupy. Unfortunately some GUIs are rather like that, being neither fast enough nor versatile enough to provide the kind of intuitive and free-flowing experience that a successful GUI can. This is encouraging for critics of GUIs, who can point to plenty of clumsy implementations. A successful GUI should enable the user to access the most useful commands quickly and easily and supply sensible defaults for any options that may be needed. That requires decisions on which commands are "most useful", what options are sensible, and how to make the commands available. Rather than discuss which statistical tools are most important (an interesting and doubtless controversial topic), this article discusses how to make objects and commands accessible in a GUI for statistical software. Related principles will apply whatever set of commands one wishes to emphasize.

An interface is not only about interacting with software by carrying out commands, it is also about being able to work with the responses the software supplies. It is a curious feature of some GUIs that they have interactive inputs, mainly using point and click, and yet have purely static outputs. Both inputs and outputs should be considered in designing an interactive system. While graphics are a natural form of output to interact with, tables and summaries of results need to be considered graphically as well and designed accordingly so that users can interact with them. This involves viewing them as structured collections of objects, as is discussed further on in Section 5.

Martin Theus has suggested a scale of interactive operations ranging from single clicks to command lines, shown in Figure 1 (Unwin, Theus, and Hofmann 2006). All these tools may be regarded as potentially part of a GUI, but they are different in the amount of control and flexibility they offer. At the right end of the scale they are just a CLI. To the left of the scale the interface tools are intuitive and fast, with the user in control, able to directly manipulate the objects, be they cases, variables, plots or models. This is the fully-fledged GUI experience. To the right of the scale the tools are more limiting and slow, with the software in control, and objects have to be addressed by name. There is a place for both. Experts would favor tools to

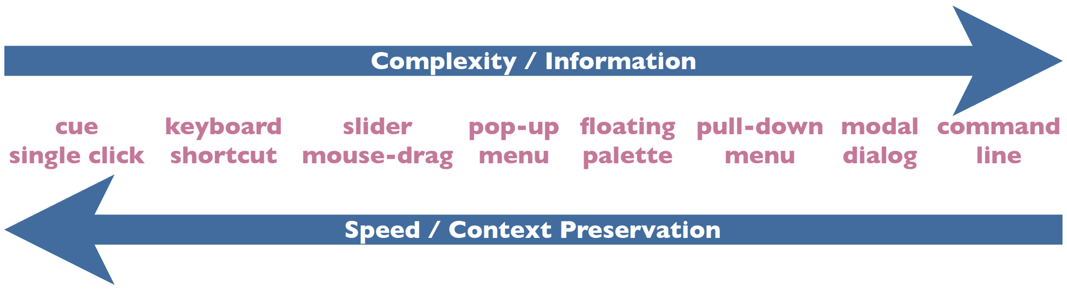

Figure 1: A scale of alternative methods for interactive commands and controls.

the left for exploratory analyses and tools to the far right for complex modelling. Beginners would favor tools to the right for taking their first steps with statistical analyses and tools to the left for working with graphics. It is not too difficult to set up GUIs for beginners, if the system is in control and only a limited set of analyses are allowed. It is a much more challenging problem to devise GUIs for exploratory analyses or complex modelling, where the user is in control and it is impossible to predict in advance what analyses might be carried out. The further you move to the right on the scale, the easier it is to provide similar, if not quite identical, interfaces on all three major operating system platforms.

Comparisons of alternative interfaces can give the impression that you must use either one or the other. Actually, it is often advantageous to offer several ways of carrying out the same operation. You can open a file by double-clicking it, using a keyboard shortcut, choosing a menu option or writing a command-line. As long as the interfaces are smoothly integrated and don't interfere with one another, this enables users to apply whatever approach is best in a particular situation. Knowing that alternatives are available takes away the stress of having to know how to do something exactly.

This paper discusses GUIs for R for exploratory analyses and refers to some of the ideas behind the **iplots** package for interactive graphics (Urbanek and Wichtrey 2011) and how it is planned to extend the package to incorporate exploratory modelling. **iplots** is very much work in progress, and until the tools are available it is difficult to say how well they will work and how they will be used. New tools lead to new possibilities and are often used in ways their designers did not expect. The paper is illustrated by looking at a dataset concerning Oscar winners and nominees and a matched control population.

3. The Oscars: Do you live longer if you are a winner?

Some ten years ago Redelmeier and Singh (2001) studied the mortality of actors and actresses who had been nominated for the Oscars. For each star, they selected a control from the corresponding film and compared the survival distributions of the following three groups: winners, nominees who did not win, and the controls. Although controls were selected in a paired way, only group comparisons were made. The paper claimed that winners live longer and this conclusion prompted a spirited discussion in the medical research literature with the general view being that the methodology was not sufficiently sound to justify the conclusion. A major criticism was that the authors did not take full enough account of the 'immortal time' bias. The dataset is available as a supplement to this paper and was originally taken from http://www.annals.org/content/145/5/361/suppl/DC1. There is information on 1670 actors and actresses, including 768 nominees and 239 winners (some people won or were nominated more than once).

4. Analyzing the data on the Oscars

Donald Norman has suggested that system design should be activity-centred rather than human-centred (Norman 2005). He has proposed a hierarchical approach from activities to tasks to actions to operations. In terms of the Oscars dataset, the main activity is analyzing the data to compare survival time distributions and individual tasks would include getting to know the data, reviewing the data quality, and building models. Individual actions might be

1. Drawing a graphic, e.g., plotting a barchart of the variable gender.
2. Transforming variables, e.g., calculating the time between first nomination and first win.
3. Modelling, e.g., adding another explanatory variable to a model.

And actions are composed of operations, which are the actual interactions with the system. These operations will either be carried out through a CLI or a GUI.

4.1. Drawing a graphic

Plotting a barchart in R would require the operations of writing an appropriate command line and setting the relevant options. (It might also be necessary to open a new graphics window.) You would have to know the name of the variable. Is it 'gender', 'Gender', 'GENDER' or sex'? In fact, it is the ill-chosen 'Male=1', which appears in R as 'Male.1'. A GUI showing a list of the variables in the dataset would be preferable here, as would having the dataset in a more readily understandable form with category names instead of numbers and sensible variable names. Of course, there are GUIs and GUIs. A common approach is to choose the action ('plot barchart') and then get a dialog window offering the list of variables and some further options. The user may additionally have to click 'OK'. It seems more natural to have the variable list always available (these are the basic objects you work with), select the ones you want and then choose the appropriate plot command. Within R the resulting simple barchart (Figure 2) was drawn with

```
R> barchart(table(Oscars$Male.1))
```

It already raises an interesting question: Why are there not the same number of men and women? Using a CLI like R you would draw up a table to discover that there were 893 men and 777 women. (Using a GUI you could query the bars in the graphic directly or, as was possible in the software **MANET** (Hofmann 2000), query the variable name in the variable list for summary information.) You would think there would be the same number of male and female winners and nominees each year. For this kind of exploratory work a GUI is probably best. Drawing barcharts of numbers of nominations per person by gender and querying the bars with zero nominations reveals that there were 505 male controls and only 397 female controls, making up almost the entire difference. Presumably it was more difficult to find female controls and the same actresses were used as controls for different nominees,

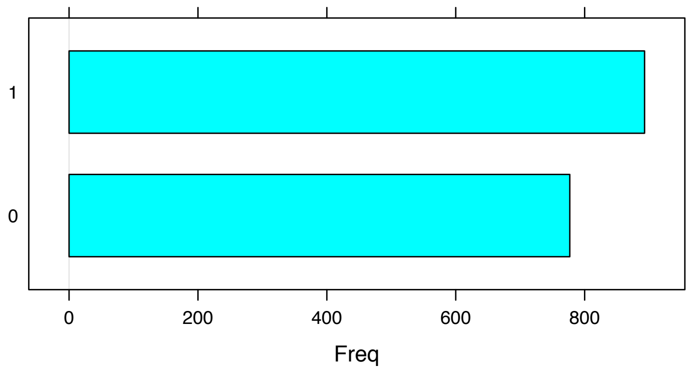

Figure 2: A barchart of numbers of males (above) and females (below) in the Oscars dataset.

but would that not stretch assumptions of independence? Since the Oscars for supporting actors and actresses only started in 1936 there should have been 138 male and female wins in all up till 2000 (73 for the main award and 65 for a supporting role). Both the main male and female awards were jointly awarded on one occasion (males in 1932, females in 1968) so each sex should have 139 wins. It turns out that there are 141 female wins (shared amongst 119 winners) and 140 male wins (shared amongst 120 winners). The reason must be that in the first year of the awards, actors could win for their performance in more than one film. Janet Gaynor's award mentioned three films and Emil Jannings' two. Although each only won one Oscar overall, the dataset reports them as having won three and two Oscars respectively. Reporting the counts in specified subsets, as we have just done, is the sort of thing database software is good at and so is R's CLI. Tracking down the reasons for differences in counts is an exploratory activity and in this case used a combination of the Oscars dataset, a separate dataset downloaded from the web listing the names and films of all nominees and the website IMDb. There are two steps to carry out, firstly tracking down any discrepancies and secondly discovering whether they are errors or have some plausible explanation. GUIs are much better than CLIs for this.

4.2. Transforming variables

Another data quality issue was discussed in the appendix to one of the follow-up articles on the Oscar dataset. (Sylvestre, Huszti, and Hanley 2006) spotted that there was an actress whose first nomination was recorded as being long after her first win and that there was an actor who had died the year before he was nominated. The first error can be seen in a scatterplot or by looking at a derived variable (the time between first winning and first nomination), as with

```
R> Oscars$WN <- Oscars$FirstWin_Year - Oscars$FirstNom_Year
R> summary(Oscars$WN)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's 
  -8.000    0.000    0.000    3.816    4.000   44.000 1431.000 
```

and it is easy enough to find out who the actress was, and where the error lay from the web, by using the other information for the case (Shirley Jones had one win in 1960 and was born in 1934. The year of first nomination should have been 1960, the year of her win, not 1952). Finding the necessary information is simpler with an interactive GUI query than with a listing of the data in R, though there is not a lot of difference. The second "error" was not an error. The actor, Massimo Troisi, who starred in the Italian film "Il Postino" was indeed nominated posthumously (as were, on different occasions, various other actors and actresses, whom the article does not mention). A scatterplot does not work here for identifying the case, as the difference of only one year is hard to spot in the graphic. The reason that the other posthumous cases were not identified is that the dataset only contains the year of first nomination and not years of later nominations. It was checking on Troisi on the web that led to the information. Good interactive GUIs encourage exploratory analyses and an exploratory philosophy. Would users of CLIs go searching on the web?

Sylvestre *et al.* (2006) excluded both cases from their analyses. Of course, two cases amongst so many should make little difference. The important step is to check the quality of the data to ensure that there is not a much larger numbers of errors. Calculating functions of the variables in a dataset is probably much easier with a CLI (at least for statisticians!), though sometimes variable names in R can be long, unwieldy and not easily distinguishable. (This is why forward completion, which is available in some CLIs is not always as helpful as it might be.) A possible advantage of GUIs here lies in being able to drag and drop variables into formulae, while the disadvantage lies in setting up the formula in the first place. A combined approach where you set up the formulae in a CLI and drop the variable names in might be best.

4.3. Modelling

During modelling you often want to add additional explanatory variables to see how much they improve the model. With a CLI you may have an `add(model, term)` command or you can copy the command for the previous model and type in an extra term. An intuitive approach for a GUI would be to have the model available as an object to which the variable could be added. This would work nicely for the addition of a simple linear term and **DataDesk** offered it as long ago as the 1980's. It would not be so straightforward for adding something more complicated, such as an interaction. Other GUI software packages provide dialog boxes giving the detailed control necessary, while offering so many options that it is hard to keep track of all the possibilities. Within R, an analyst knows that all parts of the model are accessible to him and that there will be commands which will enable him to carry out whatever modelling action he has in mind, he just has to track down the object and choose the command (which may be available in several similar, though not identical, ways). For a regular user this is not an issue, for a casual user it can be frustrating. This example illustrates a weakness of interfaces, whether CLIs or GUIs, which has nothing to do with the interfaces themselves: The underlying actions which the interface operations allow the user to carry out have to be well structured. No amount of interface design, however good, can provide an effective interface to a jumble of actions. Returning to Norman's proposed hierarchy mentioned at the beginning of the section, actions need to be grouped sensibly into tasks for carrying out particular activities.

For the survival analysis of the Oscar dataset both Kaplan-Meier and Cox proportional hazards models have been used. Code such as the following (after transformations to derive

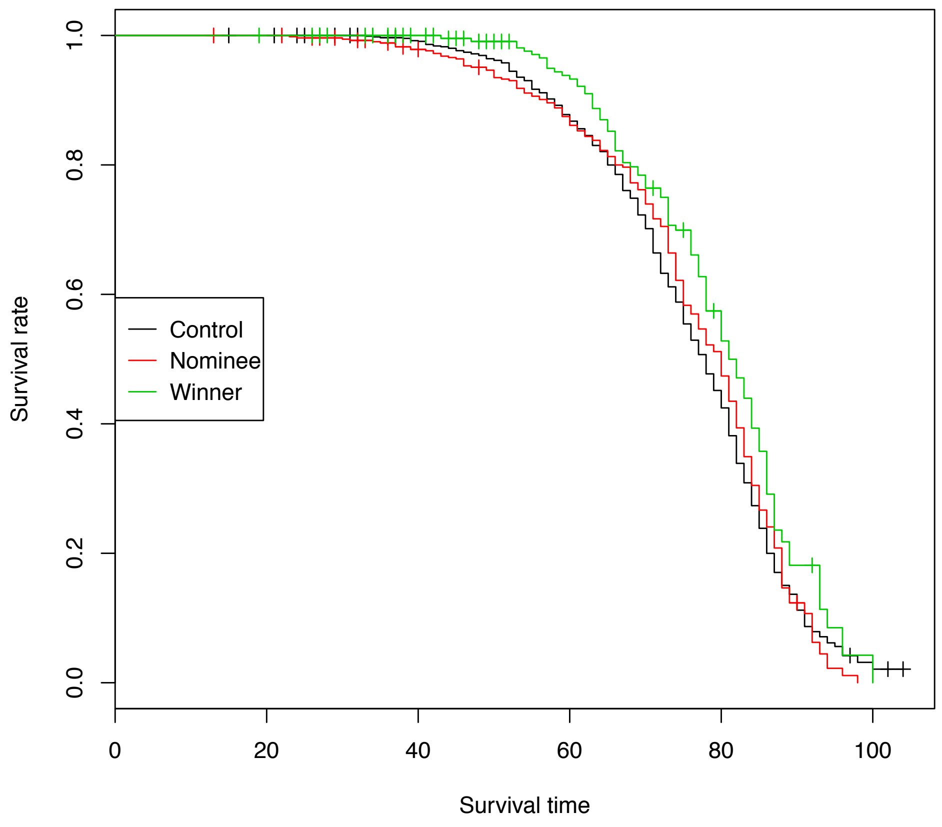

Figure 3: Survival rates for Oscar winners, for nominees who never won, and for the controls.

the required variables) would carry out the analyses (results not shown here) and produce a Kaplan-Meier plot (Figure 3) using the **survival** package (Therneau 2012):

```
R> c2 <- coxph(Surv(Oscars$stime, 1 - Oscars$Alive) ~ Oscars$Group)
R> summary(c2)
R> c2s <- survfit(Surv(Oscars$stime, 1 - Oscars$Alive) ~ Oscars$Group)
R> plot(c2s, col = 1:3, xlab=  "Survival time", ylab = "Survival rate")
R> legend("left", legend = c("Control", "Nominee", "Winner"),
+    lty = 1, col = 1:3)
```

There are several ways a GUI might usefully interact with this plot. You could query individual values, add information on group sizes or draw some kind of interval, zoom in, even redraw the plot to emphasize differences. Model results would offer opportunities for interaction too. You could change the model form or add or delete a term. Perhaps you could limit the model to a subset of the data. You could examine the residuals in various ways or display confidence intervals for the parameters. **JMP** offers many options on its survival plots via pop-up menus. Its model outputs offer a range of different possibilities, which you can display or not, as you think fit, by toggling button controls. There are also various options on the pop-up menus.

The output from the R analyses carried out has not been shown here, because it does not include all the results you might potentially be interested in. For these purposes the output from the command `str(c2)` would be more relevant, listing all the components of the results object, but that would then be far too long.

Textbooks may occasionally leave you with the impression that to fit a model to data you only have to do just that. You may check residual plots and tinker a little with your model, but the model you fit is the end result. In practice modelling is much more complicated, and there may be many different models that might be taken into consideration. A major task in any modelling process is therefore to compare the results of all the models that have been fitted. Surprisingly, this is not a task that software automatically supports. Analysts have to pull together results of different models themselves, perhaps to compare the models on several criteria, perhaps to compare the patterns of residuals, perhaps to compare coefficient estimates. A CLI like R's is effective here, as related components in different models have the same name. However, a GUI would be advantageous for listing the models and dragging them into a comparison display. Can any user ever remember what the difference was between the models they called `m1` and `m2`?

5. Structuring the interface

Early statistical software produced results one at a time and it was not easy to use the results of one procedure as input to the next. As software has progressed, this situation has changed very much for the better and it is possible in R to not only access results, but also the intermediate steps of a procedure. Basically everything is an object and the main kinds of object that we want to interact with may be classified as follows:

Cases The properties of a case are its attributes (the case values on the variables). Cases may be queried, selected, grouped or filtered to form subsets of interest.

Variables The properties of a variable are summed up in various statistics, though they also include scaling and other meta-information. Variables may be summarized, plotted, transformed, included in models or otherwise analysed.

Plots Plots have many components, some which are more statistical in nature (e.g., the bars of a histogram, the aspect ratio, the axes) and some more presentational (e.g, the background color, the size of tickmarks). Plots may be amended and varied. They are by definition linked to variables and cases, and they may be further linked to other plots or to models via cases and variables. Selecting and highlighting cases in one plot can lead to those cases being highlighted in all other plots. Reordering the categories of a variable in a barchart can lead to the same reordering of those categories where the variable is part of mosaicplots. Changing a model should be reflected in changes in the residual plots of the model. (These examples refer to interactive systems with dynamic updating.)

Models Models have many components, those which define the form of the model, those that are produced as part of the model-fitting process (e.g., a design matrix), and those that comprise the model results. The number and variety of components that are associated even with a simple linear model can be seen by applying the command `str()`. Model residuals may be analysed, predictions may be made with the models, models may be compared with one another. (Algorithmic procedures such as cluster analyses are included in this class.)

Few classifications are unequivocal. Some models are closely associated with plots (for example, a smoother in a scatterplot) or with variables (for instance, a density estimate). Scales are especially interesting. Are they associated with a variable (so that we might have the same scale for each plot a variable appears in) or are they associated with a plot axis (we may wish to use the same scaling for all variables in a parallel coordinate plot or for both variables in a scatterplot or for a group of plots of a similar type)? We might reasonably argue that they are a kind of model, something inbetween variables and models.

Interacting with cases and plots is well-known and has been the basis of interactive graphics software with GUIs, as exemplified by commercial packages like **DataDesk** and **JMP** and research packages like **GGobi** (Cook and Swayne 2007) and the Augsburg Impressionist packages (Theus, Hofmann, Gribov, and others 2010). Interacting with graphical displays is a natural operation, the objects you want to work with are visible in front of you. Cases are represented individually as points or aggregated in bars or rectangles, so that they can queried, selected or reformatted. Plots can be grown, rescaled or zoomed. If variables are presented in their own window as named icons (as in **DataDesk**) or in a list of names (as in **Mondrian**), then they can be readily accessed. (It is one of the minor irritations in using R that you often have to remind yourself of the exact names of the variables, as there is no such aide mémoire list to hand.) There is one caveat, though. If there are a lot of variables, then all interfaces will have difficulties keeping track of them.

For models the situation is different, as there are many different kinds of models and many different possible follow-up analyses. R's output tends to be rather terse, so the objects you might want to work on are not visible. **JMP** supplies more detailed outputs, and many possible options are available through a variety of pop-up menus. In R's CLI you have to know which command you need to use to attain your goal, whereas in **JMP**'s GUI you have to know in which menu you can find the kind of command you need. The sheer range of possible alternatives gives any kind of interface problems (though you might ask yourself if all these alternatives are really worth having). Working with the results of algorithmic procedures brings different kinds of challenges. The results of hierarchical clusterings and tree methods can be displayed graphically, so that they could be worked with interactively. Simon Urbanek's **KLIMT** (Urbanek 2006) demonstrates how it can be done with trees. Not all procedures lend themselves to such an approach, though that is probably the best way forward, to represent results visually so that the objects you might want to work with are directly available.

Martin Theus's **Mondrian** (Theus and Urbanek 2007) has some limited links to R via **Rserve** (Urbanek 2003) and can add interactive density estimates to histograms and interactive smoothers to scatterplots by having R carry out the calculations. However, it is not able to take full advantage of R's power and has quite a different kind of interface. In particular, **Mondrian** was designed as a closed system to provide powerful intuitive tools at the expense of not offering any easy way for users to extend them. Closed systems can be consistent and the interface can be fine-tuned, whereas open systems allow users great freedom not to be consistent (insert your own list of inconsistencies of R packages here) and there is little point in trying to fine-tune an interface that can be changed at will by the users. The initial version of **iplots** was an attempt to provide the same interactive graphics as **Mondrian**, but linked closely to R, and to provide the opportunity for others to develop their own interactive plots. One major hurdle was the need for the interface to work in the same way on all three common operating systems, Windows, Unix and Mac. Each of them has their own way of doing things and this hampers the implementation of GUIs working exactly the same way on each platform, so the interface ended up being predominantly menu-driven without an attractive GUI and the package was actually rather detached from R. This experience emphasized the need for another approach, to tie **iplots** more closely to R, to extend it to incorporate modelling as well as graphics, and to develop a real GUI for interacting with these capabilities.

For interactive graphics, the structure a GUI should have is obvious. Variables need to be selectable as objects so that you can plot them, and statistical components of graphics have to be selectable so that you can work with them, changing formats or scales, and linking cases across plots. For statistical analyses something similar applies, only here the components of the results of analyses have to be selectable. R provides almost everything as an object which can be accessed via a CLI. For a GUI these objects have to be made available and accessible visibly.

6. GUI principles

6.1. Everything is an object

Perhaps this idea could be better expressed by saying that everything you want to work with has to be an object. That is actually true for all interfaces, not just for GUIs. With a CLI you identify an object by its name or by its location. With a GUI you identify an object by one of its representations (possibly including a name or location). Ensuring that everything is an object, not just plots, but statistical components of plots, not just models, but all the inputs, options and outputs associated with the models, guarantees that you can carry out all the analyses you might want to. There are two caveats that must be noted: The dataset may not contain the data you need for your planned analysis or the dataset may not be in the right form for it. In the first case you have to go looking for more or better data and in the second you have to rearrange your dataset.

6.2. The noun-verb paradigm

A GUI has to show you what you can do in a given situation or to make it apparent in some way what you can do. Objects can be visible and you can think about what you might want to do with them. Alternatively, commands may be listed and then you can think about which command you want to apply to which object. There was some discussion in the 1980's about which approach was better. The Macintosh operating system worked primarily with the noun-verb paradigm, so you selected a file, say, and then decided if you wanted to open, copy or print it. Other systems worked with the verb-noun paradigm, so you decided to, say, open something and then chose the file you wanted to open. For exploratory analyses, and probably for other forms of analysis as well, the noun-verb paradigm is better (Raskin 2000). The example of plotting a barchart was discussed in Section 4.1. If there are many alternative objects (e.g., if you have a hundred data files), then you need to search to find the one you want to work on first. For searching amongst a large number of alternatives the verb-noun paradigm is used.

6.3. The importance of design

The reputation of GUIs is tarnished by users' experiences with poorly designed ones. There are several common faults that can be found: overburdened menus, fussy dialog boxes, unintelligible icons, and crowded screens. These flaws are due to bad design, not because they are parts of GUIs. Simply adding all sorts of possible commands to menus (sometimes in distinctly idiosyncratic orders and arrangements) does not make for a good interface. Requiring users to fill out dialog boxes that resemble bureaucratic forms more than supports for statistical analysis would make anyone yearn for a simple command line. Representing commands by small individual symbols that are difficult to decipher can be more irritating than helpful. Putting all the commands on screen at the same time is not constructive. In principle they are available, but they clutter the screen, take away screen real estate, and distract from the task at hand. Needless to say – but worth emphasizing – you can find software with straightforward and powerful menu systems, disarmingly simple dialog boxes, immediately recognisable icons, and elegantly laid out screens. It's just that you won't find these features in all software GUIs. Returning to the scale of interactive methods in Figure 1, the more direct and immediate you can make a command, the more it is to the left of the scale, the more that operation offers the advantages associated with an effective GUI. Unwin and Hofmann (1999) discussed principles of good design for statistical software over ten years ago and that discussion is as valid now as then. Consistency, immediacy, feedback, providing undo capabilities, making low demands on users' memories are all important.

6.4. Experts rather than beginners

Perhaps software developers are thinking too much about beginners and casual users. If you don't know what to do, then having the possibilities laid out in front of you can be of considerable help. The trouble is that you still have to decide between those many options. There are similarities with stereo sound systems. Some systems have controls taking up all available space, even though many of these controls will never, or hardly ever, be used. The most expensive systems appear to have no controls, everything is hidden, although you know the controls are available, an example of so-called affordance (Norman 1988). The best design makes it clear what is available, even if it is not directly visible. For some tasks there may be no way of implying how they can be carried out without displaying the command; for many tasks there are more subtle, though equally effective, ways of doing it. Sometimes these rely on analogies (moving a slider will change the value), sometimes on conventions (double-clicking a file on a Mac will have the file opened by the appropriate application). Experts and regular users prefer systems they can use without excessive guidance, however well-intended.

6.5. Styles of working

A crucial difference between CLIs and GUIs lies in the style of working. CLIs encourage a linear approach, following one goal at a time. GUIs use multiple windows and encourage a (semi-)multitasking approach. It is easy in a good GUI to select several variables and produce many plots at once, while this is not a simple task in most CLIs. Plots in a fully-fledged GUI can be linked and analysed simultaneously, with direct querying and adaptable selecting. Plots in a CLI can be examined and then, if it seems useful, redrawn. Modelling is different again. Sometimes you may have a model and want to check some minor adjustments or choose the next step from a few clear-cut alternatives. A GUI which allows you to interactively flip between alternatives exploring the possibilities is more direct and efficient than building a series of models, which would be the CLI way of doing things. On the other hand, if you want to keep a careful record of everything you have done and make some detailed comparisons between fairly similar models, both interfaces are fine, though if you know in advance which models you want to fit, then that could possibly be efficiently packed into a macro in a CLI and could easily be rerun with another dataset of the same type. To a great extent, which approach one prefers is a matter of personal taste and a matter of tasks to be carried out. Some activities are better served by one (exploratory analyses should be flexible and open-ended, using lots of graphics) and some by the other (standard analyses should be carried out in a reproducible step-by-step manner).

7. Advantages and disadvantages of GUIs and CLIs

As well as discussing the pros and cons of different approaches in general terms, it is helpful to look at additional specific examples:

- Histogram binwidths
  There are several theoretical works on choosing 'optimal' binwidths for histograms and you can specify which you want to use in a CLI command. With an interactive GUI you could vary the binwidth interactively. Neither is necessarily the best approach here, as often the meaning of the variable determines the appropriate binwidth. For histograms of the year of birth, the number of fourstar films made or the year of first being nominated in the Oscars dataset, it is evident that any binwidth chosen has to be a multiple of 1. A pop-up menu is a good interface option to control this. Should you for some reason want a histogram with unequal binwidths, something that textbooks usually discuss in theory and hardly anyone uses in practice, then a CLI would be the right approach.

- Advanced selections
  Forming a subset of all female nominees with more than three nominations and at least one win who were born before 1960 is the sort of query databases are designed for and if you can write SQL queries this may seem rather simple. However, the command gives you little feedback on which of the conditions is most restrictive and what would happen if you changed one, say requiring more than four nominations or considering the corresponding male group. As part of checking data quality it turned out to be useful to find out how many were in groups like this and, if there were only a few, who they were. Selection sequences, which Martin Theus implemented in **Mondrian**, are a very neat way of carrying out the query step by step on graphical displays of the variables and being able to change any of the steps at any time interactively to assess their effect. In Figure 4 females were selected in the top left barchart, then four or more nominations in the top right barchart, then one or more wins in the middle barchart and finally all born before 1960 in the histogram. The last three selections were all made as intersection selections. This is a prime example of a direct interactive GUI tool. You could switch to males by moving the selection rectangle in the gender barchart, you could consider only five nominations or more by shrinking the rectangle in the nominations barchart, or you could change the age range by adjusting the rectangle in the histogram. When making these changes, the other selection criteria remain fixed. Selection sequences are

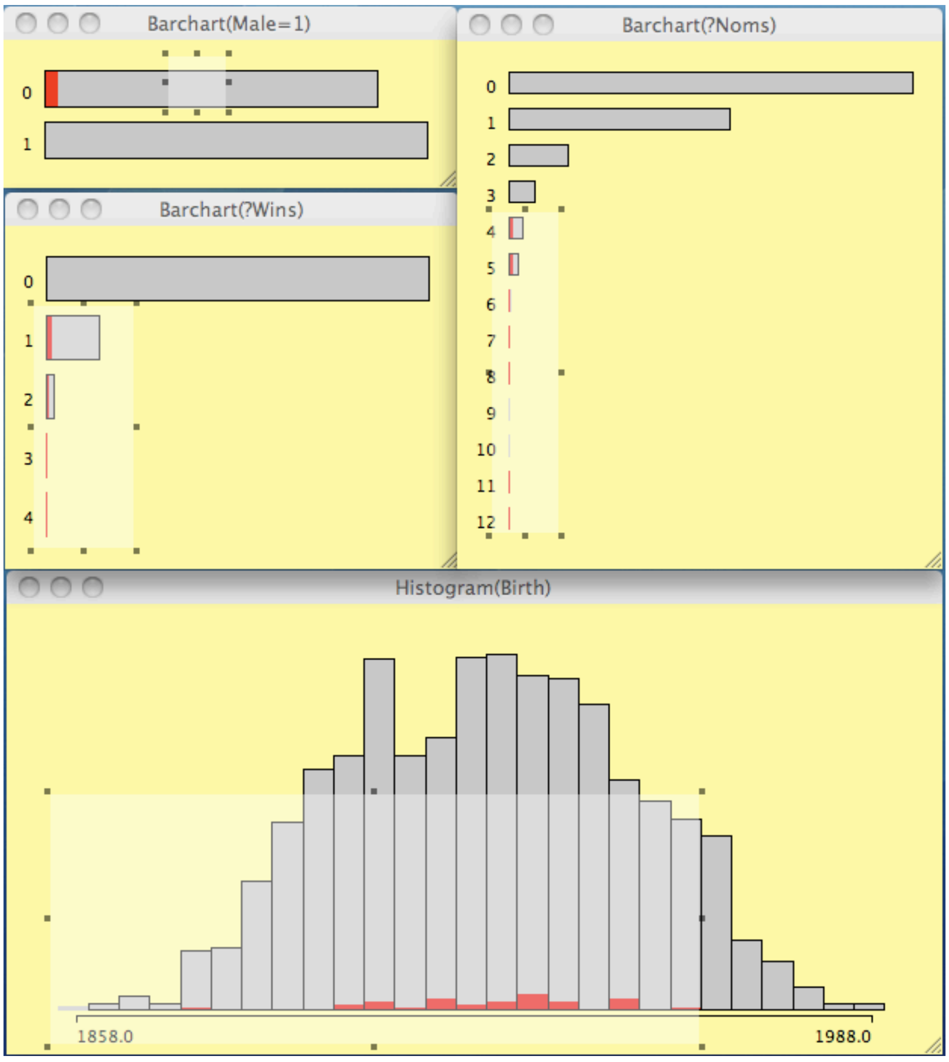

Figure 4: The selection of all female nominees with more than three nominations and at least one win who were born before 1960 using a selection sequence in **Mondrian**. The lighter rectangles are the selection areas and may be grown or shrunk using the eight 'handles', the black dots. The whole area may be moved by the mouse and the mode of selection changed through options in a pop-up menu. Selection sequences give great flexibility for adjusting search criteria to explore the data.

a sophisticated tool of considerable power, though they are best for one-off exploratory analyses, while for repeated applications a CLI would be preferred.

- Reordering bars in a barchart
  Barcharts are often drawn with the categories in alphabetic order or using some other rule that has nothing to do with the data. It can be helpful to order by count, by proportion selected or by number selected. Sometimes reordering by hand to placing the bars for two categories beside one another is valuable. Doing this sort of operation with a CLI is hard work, slow and counter-intuitive. With a pop-up menu in a GUI or with the capability to pick up bars and move them (both available in **Mondrian**), it

becomes straightforward and easy to understand. One of the most useful applications of this is at first sight surprising. If you want to identify a number of cases by name in a large dataset, select them, draw a barchart of the name variable and sort it by number selected. This would be useful in the Oscars dataset, for instance for finding out who the thirty female actresses were, who were selected in the selection sequence, if only we had the actors' names. (Unfortunately they were not included in the dataset.) For printing out a list of the names in a report a CLI would be needed.

- Querying cases or parts of plots, selecting cases, zooming in in plots are further actions where GUIs have the advantage. Zooming is an instructive example. For zooming in on "the points close to zero", a GUI is better. For zooming in precisely on the square of side 2 around zero, a CLI is essential.

Of course, all these examples are associated with exploratory analyses. More formal actions, like transforming variables or model-building, which both amount to writing down equations, are better served by CLIs for constructing or building, but by GUIs for examining the results.

8. One last look at the Oscars

The Oscars dataset is not unusual is being both surprisingly comprehensive and yet disappointingly incomplete. Information on the pairings could be valuable, as would other information which would enable more sophisticated survival models. In the spirit of this paper, it would be attractive to have a summary of the various models of the data in an interactive table, permitting us to compare and contrast the results in various ways. This naturally presupposes that we are all in agreement about the data. In some further exploratory analysis, two new plots were drawn side by side (Figure 5) using the following code:

```
R> boxplot(X.FourStar_Films ~ X.Noms, data = Oscars)
R> plot(Oscars$X.Noms, Oscars$X.FourStar_Films, pch = 16)
```

Any films receiving four (or five stars) in the All Movie Guide (allmovie.com) were considered to be of higher quality than the rest and so the number of fourstar films an actor was in was regarded as a better measure than their total number of films. In the display of parallel boxplots on the left, we can see as we might expect that actors with more nominations have been in more 'good' films. The outlier on 8 nominations stands out as someone who was nominated very often given how rarely they were in a 'good' film. In the parallel dotplots on the right, the actor with 4 nominations, yet no 'good' films, catches the eye. There is nothing more that can be done with these static plots. Using **Mondrian** (or **iplots**) it is possible to select these cases and either link them to other displays for further case information or query them. It turns out that both were actresses who each won one Oscar. Despite her 8 Oscar nominations Geraldine Page did indeed appear in only 4 fourstar films, but the data on Joanne Woodward's fourstar films is just wrong. The All Movie Guide suggests a figure of 10 rather than 0, so perhaps it was a data entry slip. Both this error and the ones discussed earlier were easy to track down with a GUI and interactive graphics. Fortunately, it appears that these are not major errors, but these examples do show how useful graphics and GUIs can be in checking data quality. Powerful model-building tools are invaluable – as long as the data are sound.

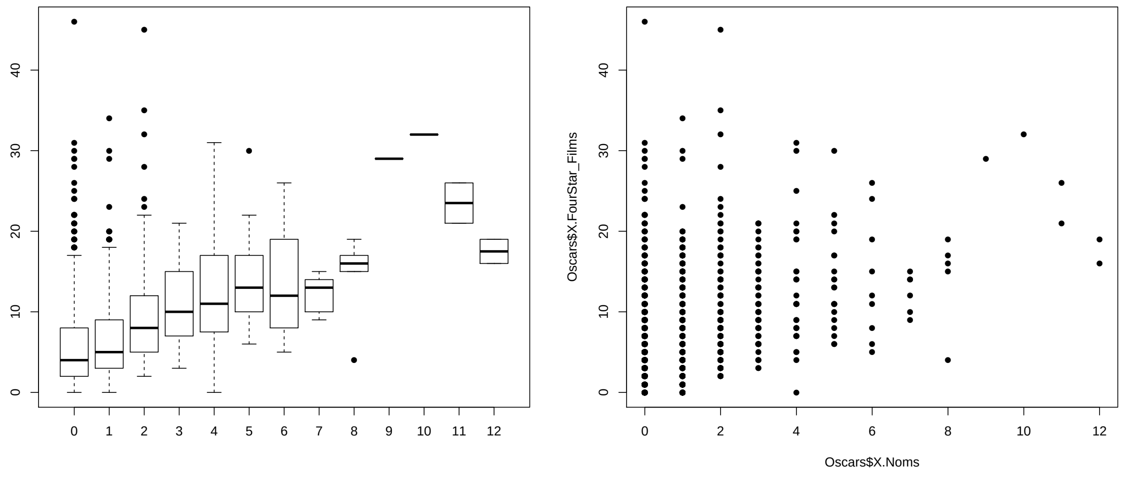

Figure 5: Numbers of fourstar films by numbers of nominations.

The value of GUIs for exploratory analyses has been a common theme throughout the article. Two interesting features of the Oscars dataset found in this way are that more winners changed their names than nominees than controls and that of the 42 winners who were younger than 30 when they first won, X were women. Now do you think X is big or small? And how big or small? (Hint: The number appears in the title of a classic Hitchcock film.) More seriously, this difference in prize-winning age distributions coupled with the large difference by sex in the numbers of controls, observed in the discussion of Figure 2, suggests that it might be better to analyse the data separately for the sexes. Sex is included as a factor in Redelmeier and Singh's original paper and they say it does not affect their conclusion regarding the effect of winning on survival. Repeating their analysis confirms their conclusion for their model, but analyzing the data in two models, one for men and one for women, leads to a different conclusion: The effect is no longer significant for men and less significant than before for women. It may be ingenuous to believe that including sex as a factor in a model can incorporate all the differences between the sexes that affect the results.

9. Combining interfaces for R

An essential feature of any interface is consistency, what you could call the power of conventions, and this is where R and similar cooperative projects have their problems. It is impossible to ensure that each R package follows the same interface standards. There are some general rules in writing for R (for instance, in the way commands are formed and how help files are structured) and some commonly agreed principles (as in the way models are specified), but that still leaves much room for variation. A fast and flexible interface is only possible if the user can have confidence that what works in one situation will work in another. Imagine driving a car and not knowing as you went round a corner into another street whether people drove there on the left or on the right or, indeed, down the middle. It would slow you down (hopefully!) and give you much to think and worry about that had little to do with your task in hand. This is one of the reasons GUIs in R are destined to be limited in the range of commands they can offer. Another reason is the restricted number of genuinely interactive operations that can be considered. Recent developments with tablet computers have extended the possibilities, but we still have only a small number available.

The lack of a sufficient number of direct operations can be circumvented to some extent by working more with command combinations. Instead of looking for a single complex command which does everything at once, it may be more appropriate to combine several simple commands. This can be both more transparent and easier to remember. You have to balance the time and effort needed to carry out several elementary commands against the time it may take to work out how to achieve the same goal in one step.

The kind of interface you have available will influence the kind of analyses you are likely to carry out. If you see some outliers in a scatterplot, it is an obvious next step with a GUI to query them or select them. With a CLI, you would have to think hard about how to identify the points and you might decide to pass on that analysis. Similarly, if there is a cluster of points that arouses your attention you might zoom in with a GUI to take a closer look, while redrawing the graphic with a CLI (maybe more than once till you get the right zoom level) would involve more work and be a less attractive option. For transforming or combining variables a CLI is the intuitive approach for those used to writing mathematical formulae and so a CLI is better for statisticians. If there are several explanatory variables in a dataset, then an analyst with a GUI might investigate the effects of individual variables graphically first, both to get a feel for the data and to see what main effects there might be. Multivariate analyses could be trickier with a GUI, whereas with a CLI it would be easy to fit a full model straight off and see which variables seem to play an important role. The advantage of reproducibility of a CLI is mitigated by the lack of feedback, which makes it more error-prone. With GUIs you can generally see what you are doing and amend as you go. Although new ideas and substantial improvements can be expected in GUIs, CLIs will improve as well. It is often surprising how small amendments can improve the users' experience. An integration of both approaches would be the ideal.

The ideas presented in this paper are part of the discussion taking place about the future development of **iplots**. Simon Urbanek has rewritten the engine of the package to make it much faster and more powerful (Urbanek 2009). An interface similar to that of the original **iplots** package would enable the use of interactive graphics. An additional programming interface would allow expert users to extend the package with their own graphics. How models from R can be incorporated is the key question. If R model objects and their components can be given a more integrated structure, allowing easier navigation while still taking full account of the models' analytic complexity, then a GUI could be designed to give access to them. Without this underlying structure GUIs are condemned to limited ranges of possibilities – however successfully and attractively they work within their particular range.

Acknowledgments

Thanks to Alex Gribov, Heike Hofmann, Simon Urbanek, and Martin Theus for their software and for valuable discussions. Thanks also to Redelmeier and Singh for collecting the Oscars dataset and for making it available to others for analysis. And thanks to the referees for their constructive suggestions.

References

Cook D, Swayne D (2007). *Interactive and Dynamic Graphics for Data Analysis.* Springer-Verlag.

Grosjean P (2010). "R GUI Projects." URL http://www.R-project.org/GUI.

Hofmann H (2000). "**MANET**." URL http://stats.math.uni-augsburg.de/MANET/.

Norman DA (1988). *The Design of Everyday Things.* MIT Press.

Norman DA (2005). "Human-Centered Design Considered Harmful." *Interactions*, **12**(4), 14–19.

Raskin J (2000). *The Humane Interface: New Directions for Designing Interactive Systems.* Addison-Wesley.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Redelmeier DA, Singh SM (2001). "Survival in Academy Award Winning Actors and Actresses." *Annals of Internal Medicine*, **134**, 955–962.

SAS Institute Inc (2012). "**JMP** 10: Statistical Discovery Software." URL http://www.jmp.com/.

Sylvestre MP, Huszti E, Hanley JA (2006). "Do Oscar Winners Live Longer than Less Successful Peers? A Reanalysis of the Evidence." *Annals of Internal Medicine*, **145**, 361–363.

Therneau T (2012). ***survival**: A Package for Survival Analysis in S.* R package version 2.36-12, URL http://CRAN.R-project.org/package=survival.

Theus M, Hofmann H, Gribov A, others (2010). "Augsburg Impressionists." URL http://rosuda.org/software/.

Theus M, Urbanek S (2007). *Interactive Graphics for Data Analysis.* CRC Press, London.

Unwin AR, Hofmann H (1999). "GUI and Command-Line — Conflict or Synergy?" In K Berk, M Pourahmadi (eds.), *Computing Science and Statistics, Proceedings of the 31st Symposium on the Interface*, volume 31, pp. 246–253. Interface Foundation, Chicago.

Unwin AR, Theus M, Hofmann H (2006). *Graphics of Large Datasets.* Springer-Verlag, New York.

Urbanek S (2003). "**Rserve** – A Fast Way to Provide R Functionality to Applications." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003).* Vienna, Austria. ISSN 1609-395X. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/.

Urbanek S (2006). *Exploratory Model Analysis.* Books on Demand.

Urbanek S (2009). "iPlots eXtreme." In *useR! 2009 – The R User Conference*. Rennes. URL http://www.R-project.org/conferences/useR-2009/slides/Urbanek.pdf.

Urbanek S, Wichtrey T (2011). ***iplots**: Interactive Graphics for R.* R package version 1.1-4, URL http://CRAN.R-project.org/package=iplots.

Velleman PF (1997). "**DataDesk** Version 6.0 – Statistics Guide."

Affiliation:

Antony Unwin
Department of Computeroriented Statistics and Data Analysis
University of Augsburg
86135 Augsburg, Germany
E-mail: unwin@math.uni-augsburg.de
URL: http://rosdua.org/~unwin/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume 49, Issue 11 *Submitted:* 2010-12-08
June 2012 *Accepted:* 2011-07-15